Разработка на тема 2:

1. Сорс контрол - система за контрол на версиите, като **SVN, GIT, GITHUB**

Контрол на версиите (Version Control) ≈ управление на софтуерни конфигурации (Software Configuration Management / SCM)

* + Една от дисциплините в софтуерното инженерство
  + Съдържа техники, практики и инструменти за работа със споделени файлове и програмен код
  + Има механизми за управление, контрол и проследяване на промените
  + Дефинира процеса на управление на промените
  + Описва какво се е случило в проекта с течение на времето
  + Разрешава конфликтите, възникнали при промените



Използваме системите за контрол на версиите за улесняване на работата в екип

* + Кодът се съхранява в централно хранилище
  + Води се опис на всички промени в проекта
  + Лесно се разрешават конфликтите, възникнали при сливане на промени

**Git == разпределена система за контрол на програмния код (source-control system)**

* + Най-популярната в света (към момента)
  + Свободна, софтуер с отворен код
* Работи с локални и отдалечени хранилища
* **Git bash** – команден интерфейс към Git

**Git команди**:

* Клониране на съществуващо Git хранилище

**git clone [отдалечен url]**

* Изтегляне и сливане на промени от отдалечено хранилище

**git pull**

* Подготовка (добавяне / избор) на файлове за запис

**git add [файл] ("git add ." добавя всичко)**

* Предаване (commit) към локалното хранилище

**git commit –m "[вашето съобщение]"**

**GitHub** е номер едно сайт в света за хостване на програмен код

* + Безплатен за проекти с отворен код
  + Платени планове за частни хранилища
* GitHub предоставя:
  + Git хранилище за програмен код
  + Тракер на задачи (бъг тракер)
  + Табло на проекта (тип [Канбан](https://bg.wikipedia.org/wiki/%D0%9A%D0%B0%D0%BD%D0%B1%D0%B0%D0%BD))
  + Уики страници (документация)

Системата за контрол на версиите се развива през годините, като в началото ѝ стои **локалната система за контрол на версиите** (като RCS), при която файловете са били на локалния компютър на разработчика и само той е работил с тях.

**Централизирана система за контрол на версиите** (CVS, Subversion), която позволява много потребители да работят по един проект. Проектът се намира на един централен компютър, с който те осъществяват връзка.

**Децентрализирана система за контрол на версиите** (Git), която решава най-големия недостатък на другите две – в случай на повреда в централния или локалния компютър, всички данни на проекта могат да бъдат загубени, тъй като се намират на едно физическо място. **При децентрализираната система за контрол на версиите проектът се сваля и копира локално на компютъра на всеки участващ потребител, като така се създава бекъп.** В случай на необходимост, проектът може да се възстанови от локалното копие на някой от потребителите.

1. **Дебъгване**
2. **Типове данни:**
   * Стойности които имат сходни характеристики
   * Описва вида на информацията, който се пази в компютърната памет (съответно в променливата)

* **Типът данни притежава:**
  + Име (C# ключова дума or .NET тип)
  + Размер (колко памет се използва)
  + Стойност по подразбиране

**Целочислени типове данни:**

* + **Съдържат числа**
    - Имат определени диапазони
    - Могат да пазят или да не пазят знак

**Пример с целочислен тип:**

byte centuries = 20; // Много малко число (до 255)

ushort years = 2000; // Малко число (до 32767)

uint days = 730484; // Голямо число (до 4.3 млрд.)

ulong hours = 17531616; // Много голямо число (до 18.4\*10^18)

Console.WriteLine(  
 "{0} centuries = {1} years = {2} days = {3} hours.",  
 centuries, years, days, hours);

**Реални типове данни:**

Типовете с плаваща запетая:

▪ Съдържат реални числа, например: 1.25, -0.38

▪ Имат диапазон и точност според използваната памет

▪ Понякога се наблюдават аномалии при изчисления

▪ Могат да пазят много малки и много големи стойности като 0.00000000000001 и 10000000000000000000000000000.0

Типовете с плаваща запетая са:

▪ float

▪ 32-битов, точност 7 знака след запетаята

▪ double

▪ 64-бита, точност от 15-16 знака след запетаята

▪ Стойността по подразбиране е:

▪ 0.0F за тип **float**

▪ 0.0D за тип **double**

Реален тип с десетична точност:

* Има специален реален тип с десетична точност в C#:
  + **decimal**
    - 128-битов, с точност до 28-29 знака
  + Използва се за финансови изчисления
  + Почти няма грешки при закръгляне
  + Почти няма загуба на точност
* Стойността по подразбиране за **decimal** е:
  + **0.0M** (**M** е наставката за десетичните числа)

**Преобразуване на типове:**

Променливите съдържат стойности от даден тип

* Типът може да се промени (преобразува) към друг тип
  + Скрито преобразуване на тип (без загуби): променлива от по-голям тип (пр. **double**) взема по-малка стойност (пр. **float**)

float heightInMeters = 1.74f;

double maxHeight = heightInMeters; //Скрито преобразуване

* + Явно преобразуване (със загуба) – може да загубим точност:

double size = 3.14;

int intSize = (int) size; //Явно преобразуване 🡪 3

**Обектен тип в C#**

* Специален тип – родител на всички други типове в .NET
* Задава се чрез object ключова дума
* Може да приема стойности, от който и да е тип
* Референтен тип – съдържа указател към област в паметта, на която се съхранява неговата стойност

**Бройни системи**

* + **Двоична, десетична, шестнадесетична**
  + Числата се използват **за броене**, **за количествена мярка** и **имат различен запис в различните позиционни бройни системи**
    - Имат **един и същи** смисъл –  
      представляват един и същи  
      брой, едно и също количество
* Десетична бройна система

Десетични числа (**основа 10**)

**Представяни, чрез 10 цифри:**   
0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Всяка позиция представлява умножение по 10:

401 = 4\*102 + 0\*101 + 1\*100 = 400 + 1

130 = 1\*102 + 3\*101 + 0\*100 = 100 + 30

* Двоична бройна система

Двоичните числа се преставят като последователност от битове

**Най-малката единица информация** – 0 или 1

Битовете лесно се представят в електрониката

Двоични числа (**основа 2**)

* + Представя се с 2 цифри: 0 and 1

Всяка позиция **преставлява умножение по 2**:

101b = 1\*22 + 0\*21 + 1\*20 = 100b + 1b =  
 = 4 + 1 = 5

110b = 1\*22 + 1\*21 + 0\*20 = 100b + 10b =  
 = 4 + 2 = 6

**Преобразуване от десетична в двоична бройна система**

* Делим на две и прибавяме в обратен ред остатъците:

500/2 = 250 (0)

250/2 = 125 (0)

500d = 111110100b

125/2 = 62 (1)

62/2 = 31 (0)

31/2 = 15 (1)

15/2 = 7 (1)

7/2 = 3 (1)

3/2 = 1 (1)

1/2 = 0 (1)

* Шестнадесетична бройна система

Шестнадесетични числа (**основа 16**)

Представяни чрез 16 цифри:   
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E и F

* В програмирането обикновено се ползва префикс 0x

0 🡪 0x0 4 🡪 0x4 8 🡪 0x8 12 🡪 0xC

1 🡪 0x1 5 🡪 0x5 9 🡪 0x9 13 🡪 0xD

2 🡪 0x2 6 🡪 0x6 10 🡪 0xA 14 🡪 0xE

3 🡪 0x3 7 🡪 0x7 11 🡪 0xB 15 🡪 0xF

**Преобразуване на числа от шестнадесетична към десетична БС**

* Умножаваме всяка цифра по основата на съответния степенен показател:

FFhex = 15\*161 + 15\*160 = 240 + 15 = 255d

1. **В програмирането, масивът е множество от елементи**

* Елементите са номерирани от 0 до Length-1
* Елементите са от същия тип (намример integers – цели числа)
* Масивите имат постоянен размер(дължина) (Array.Length) – не може да бъде променяна



В **Length** се пази дъжината

(брой елементи) на масива

Масив от 5 елемента

* Създаване на масив от 10 цели числа:

int[] numbers = new int[10];

* Задаване на стойности на елементите на масива:

for (int i = 0; i < numbers.Length; i++)

numbers[i] = 1;

* Достъп до елементите на масива по индекс:

Операторът **[]** дава достъп до елементите

по index

numbers[5] = numbers[2] + numbers[7];

numbers[10] = 1; // IndexOutOfRangeException

* **Въвеждане на масиви от конзолата -** чрез for цикъл или **String.Split**()
  + Първо, въвеждаме броя на елементите length на масива:

**int n = int.Parse(Console.ReadLine());**

* + После създаваме масив с **n** на брой **елементи** и **ги** въвеждаме :

**int[] arr = new int[n];**

**for (int i = 0; i < n; i++)**

**{**

**arr[i] = int.Parse(Console.ReadLine());**

**}**

* **Въвеждане стойностите на масива на един ред**
  + Стойностите на масив могат да бъдат въведени на един ред, разделени с интервал:

**.Split(' ') разделя по интервал string и го записва в масив string[]**

**string values = Console.ReadLine();**

**string[] items = values.Split(' ');**

**int[] arr = new int[items.Length];**

**for (int i = 0; i < items.Length; i++)**

**arr[i] = int.Parse(items[i]);**

* + По кратък запис, чрез функционално програмиране:

**using System.Linq;**

**…**

**var inputLine = Console.ReadLine();**

**string[] items = inputLine.Split(' ');**

**int[] arr = items.Select(int.Parse).ToArray();**

* **Извеждане на масив на конзолата:**
  + За извеждане на елементите на масив може да се ползва цикъл **for**
  + Разделяне на елементите с **интервал** или **нов ред**

**string[] arr = {"one", "two", "three", "four", "five"};**

**// преминаваме през всички елементи на масива**

**for (int index = 0; index < arr.Length; index++)**

**{**

**// Извеждаме всеки елемент на нов ред**

**Console.WriteLine("arr[{0}] = {1}", index, arr[index]);**

**}**

**Деклариране на многомерен масив**

* Досега едномерен масив от цели числа декларирахме чрез int[], двумерен масив бихме декларирали по следния начин:

int[,] twoDimentionalArray;

* Аналогично тримерен масив бихме декларирали така:

int[, ,] threeDimentionalArray;